
Article:
Zou, Xueyi, Alexander, Robert David orcid.org/0000-0003-3818-0310 and McDermid, John Alexander orcid.org/0000-0003-4745-4272 (2016) Testing Method for Multi-UAV Conflict Resolution Using Agent-Based Simulation and Multi-Objective Search. Journal of Aerospace Information Systems. pp. 191-203. ISSN 2327-3097

https://doi.org/10.2514/1.I010412

A Testing Method for Multi-UAV Conflict Resolution Using Agent-Based Simulation and Multi-Objective Search

Xueyi Zou,[1] Rob Alexander,[2] and John McDermid[3]
University of York, York, England, YO10 5GH, UK

This is the accepted manuscript / post-print. The publisher's formatted and copy-edited version is here — http://dx.doi.org/10.2514/1.I010412

We present a new approach to testing multi-UAV conflict resolution algorithms. We have formulated the problem as a multi-objective search problem, with two objectives: finding air traffic encounters that (1) are able to reveal faults in conflict resolution algorithms, and (2) are likely to happen in the real world. Our method uses agent-based simulation and multi-objective search to automatically find encounters satisfying these objectives. It describes pairwise encounters in 3D space using a parameterized geometry representation, which allows encounters involving multiple UAVs to be generated by combining several pairwise encounters. The consequences of the encounters, given the conflict resolution algorithm, are explored using a fast-time agent-based simulator. To find encounters meeting the two objectives, we use a genetic algorithm approach. We have applied our method to test ORCA-3D, a widely-cited open source multi-UAV conflict resolution algorithm, and compared our method's performance against a plausible random testing approach. Our results show that our method can find the required encounters more efficiently than the random search. The identified safety incidents are then the starting points for understanding limitations of the conflict resolution algorithm.

[1] PhD student, Department of Computer Science, xz972@york.ac.uk.
[2] Lecturer, Department of Computer Science, rob.alexander@york.ac.uk.
[3] Professor, Department of Computer Science, john.mcdermid@york.ac.uk.

I. Introduction

UNMANNED Aerial Vehicles (UAVs) are attracting the attention of innovators and companies due to their huge potential for civilian and commercial use. Several large technology companies are testing UAVs for delivering goods, providing Internet access, etc., and they are seeking to get permits to operate UAVs beyond visual line of sight. Once such operation is allowed, manufacturers and operators will race to exploit UAVs for many applications, and future airspace is likely to be very crowded with all kinds of UAVs. Air Traffic Management (ATM) for these UAVs will be a major concern, particularly because of the increased opportunity for unsafe encounters.

To make safe operation possible, UAVs must provide a Sense and Avoid (SAA) capability. According to [1], to achieve SAA UAVs must maintain good situation awareness and be capable of *self-separation* and *collision avoidance*. These two sub-capabilities form a safety barrier of two layers. In the first layer (self-separation), UAVs autonomously plan their flight paths to resolve conflicts[4] with each other (and potentially with conventional aircraft) and maintain a defined safe separation distance. If this safe separation is predicted to be violated or the collision risk is predicted to be higher than a defined threshold, the second layer (collision avoidance) will provide evasive maneuvers for the UAVs to avoid an imminent collision. Both layers function in a similar way — each UAV uses its situation awareness to calculate and execute a change to its flight path to avoid violation of safe separations or collisions. Usually collision avoidance algorithms deal with pair-wise encounters and take no account of restoring the UAVs to their original flightpaths, while self-separation algorithms deal with two or more UAVs encounters and need to take the recovery of the intended flight path into account.

For the purpose of self-separation, a wide variety of aircraft (or robots, agents) *conflict resolution* approaches that have been proposed in the fields of air traffic management, automatic control, and mobile robotics [2-8] have the potential to be adapted for UAVs. However, given the strict safety requirements of the aviation sector, a conflict resolution algorithm cannot be accepted and deployed for UAV self-separation use without rigorous testing and evaluation.

The testing and evaluation of conflict resolution algorithms, in most of existing published research is in the form of *coverage-based test case generation* or *large-volume simulations*. Flight test is also indispensable for ensuring conflict resolution algorithms have the desired properties. However, due to its high cost, combined with the risk involved, it can only be conducted for a very limited set of encounter cases and thus it gives quite limited assurance.

[4] A conflict is a situation where two or more UAVs encounter each other and a violation of safe separation (and thus, potentially, a collision) would happen if no actions were taken.

It do, of course, tests real aircraft behavior. In contrast, simulation is more cost-effective and thus can cover a far larger part of the possible operational situations, but it is subject to limitations in the fidelity of the simulation. In this paper, we only survey related studies in the form of software testing and simulation.

For coverage-based test case generation, some studies by researchers from NASA Ames used model checking to generate test cases for testing the built-in conflict resolution algorithms of TSAFE [9] and the AutoResolver [10]. For example, in [11], Bushnell et al used Symbolic PathFinder (an extension to Java PathFinder [12] that combines symbolic execution and constraint solving) to automatically generate test cases that achieve a variety of coverage criteria including Modified Condition/Decision Coverage (MCDC) [13]. In [14], Giannakopoulou et al. developed a testing environment for generating test cases that cover the behavioral space of AutoResolver to a satisfactory degree.

Large-volume simulations, especially Monte-Carlo simulations, are typically used to evaluate the performance of conflict resolution algorithms rather than specifically to find individual faults. In [15], Paielli described a trajectory scripting language to help automatically generate large-volume simulated air traffic encounters, and ran simulations of the generated encounters to evaluate the conflict resolution algorithms of the TSAFE software. A Monte Carlo simulation approach [16] and an accelerated Monte Carlo approach [17] were used for probabilistic risk analysis of a next-generation air traffic control operational concept. Both studies derived quantitative results of the mean time between mid-air collisions, and identified important failure modes. Blom and Bakker in [18] also investigated the collision risk of an airborne self-separation concept using techniques in agent-based modeling and rare-event Monte Carlo simulation. Their results show that the specific self-separation concept they consider can safely accommodate very high en route traffic demands.

The studies listed in the previous two paragraphs may all discover faults and limitations in algorithms. This is often a side-effect of their analysis however — in assuring bug-freeness of an algorithm or collecting statistics about algorithm performance under diverse conditions, they often reveal faults. They do not, however, actively pursue specific faults, or conduct systematic stress testing to find the precise limits of algorithms.

In [19], we proposed a testing approach which combines agent-based simulation and evolutionary search that can find specific encounter situations an SAA algorithm cannot handle. That testing approach is most suitable for collision avoidance algorithms, where only two UAVs are involved in an encounter. With the introduction of civilian UAVs, the airspace will become more congested and many body encounters will become more likely to

arise in the future. In this paper we extend our previous approach to test conflict resolution algorithms which are distributed across multiple UAVs and which are supposed to resolve conflicts between them collectively. Specifically, we extend it to assess the limitations of the algorithms — we determine the simplest conflict (i.e. the most likely to happen in real world conditions) that the algorithm fails to resolve.

We are targeting the conflict resolution algorithms suitable for unmanned rotary wing aircraft (helicopters, multi-copters) which are very popular in civilian use due to the ease of controlling them. They are different from fixed wing aircraft and the most significant feature of these aircraft is that they can hover in the air, which makes them capable of dexterous maneuvers. Specifically, the conflict resolution algorithm we have used for our case study here is ORCA-3D, a widely-cited open source multi-UAV conflict resolution algorithm.

The rest of the paper is organized as follows. In Section II we formulate the problem to be solved and present our method as a testing framework. The testing framework includes two parts: *simulation* and *search*. In Section III we describe the simulation part of the testing framework, which we use to simulate behaviors of UAVs with conflict resolution algorithms on-board. In Section IV, we explain how to automatically generate multi-UAV encounters. In Section V, we describe the search part of our testing framework and detail the design of the multi-objective search method. Section VI is a case study, where we describe the experimental comparison of our method with a random search based approach to the problem in testing ORCA-3D. Section VII summarizes and concludes the paper.

II. Problem Formulation and the Testing Framework

In the UAV conflict resolution situation, we define an *incident* as the violation of safe separation. In this paper we focus on civilian UAVs, and our typical operational scenario is using UAVs to deliver parcels from a distribution center to customers' house. We therefore only concern ourselves with conflicts between UAVs, not with conflicts between UAV and manned commercial aircraft. It is noted that there is no well-accepted definition for safe separation and several metrics exist, which include distance-based metric, time-based metric and risk-based metric. The former two metrics concern the distance between two UAVs or the time left for two UAVs to collide. Risk-based metric, e.g. the one proposed in [20], defines a safe separation as a relative state between UAVs where the risk of collision is lower than an unacceptable level. In this paper, we adopt a simple distance-based metric — if the horizontal distance and the vertical distance between two UAVs are smaller than some threshold values at the same time, safe-separation is violated and an incident occurs. These two threshold values are both set to be *20m,* which is

reasonable given the properties of the UAVs studied including the operational scenario, slow speed, small size and high maneuverability. However, we do not claim that this creates an acceptable level of collision risk for a comparable real application. The approach described does not, of course, depend on the exact value chosen.

To test the capability of multi-UAV conflict resolution algorithms, we want to find counterexamples. That is, we want to find encounters where, despite the help of the conflict resolution algorithms, UAVs still experience violation of safe separation. The encounters should also be simple, so that they are very likely to happen in the real world environment. In this paper, we define the *cardinality* of an encounter to be the number of UAVs involved in this encounter. We assume that encounters with lower cardinality are more likely to happen in the real world (we recognize that other factors could be considered, but for the sake of this paper simply meeting the low-cardinality requirement is an adequate challenge).

Obviously, the problem can be solved by first randomly generating lots (millions, perhaps) of encounters with different cardinalities and then running simulations to evaluate these encounters. If an encounter leads to a violation of safe separation, this encounter will be recorded. After gathering enough (thousands, perhaps) of these encounters, we can select those encounters with the lowest cardinality (the simplest). These constitute simple counterexamples to any claim that the conflict resolution algorithms is perfect, and they can be used as the starting point for further analysis (maybe manually) of the limitations of the algorithm. In this paper, we call this a *random search approach*.

A primary drawback of this random search approach is that, to gather enough encounters that will lead to violations of safe separation, millions of random encounters may need to be generated and evaluated (if the conflict resolution algorithm is moderately good). The cost of running simulations to evaluate such a huge number of encounters is considerable.

In the hope of reducing the number of simulations we need to run, we have formulated this problem as a multi-objective search problem, where we actively search for encounters satisfying the following two objectives:

1) Be able to lead to a violation of safe separation;

2) Have low cardinality.

The two objectives are equally important to our problem. An encounter that is able to lead to an incident but is so complex that it will rarely, if ever, happen may not be insightful for analyzing the conflict resolution algorithm. Similarly, a simple encounter that causes no problem for the algorithm is useless for our purposes. These two objectives are in opposition, because the fewer UAVs involved in an encounter, the less likely it is to lead to an

incident. The two objectives are also incomparable — we cannot merge them into a single objective by allocating different weights to each. This problem therefore cannot be solved using a single objective search approach.

An encounter that satisfies the two objectives is a candidate solution to the multi-objective search problem. To make it easier to automatically generate encounters, in a simulated encounter we define an "own-ship", whose initial position and velocity are fixed at the normal values for them, at the beginning of a simulation. All the "intruders" (UAVs flying in the same space as the own-ship) are defined and generated relative to the own-ship. All intruders are generated in conflict with the own-ship[5]; they are not necessarily in conflict with each other.

In our simulations, we ignore incidents between pure intruders; we are only interested in incidents involving the own-ship. This is a simplification, but we think it is reasonable, because: (1) our main purpose is to find faults of a conflict resolution algorithm, rather than to prove an algorithm is fault-free; it is thus not fatal for us to miss some incidents, provided that we discover some other ones; (2) since the intruders are generated to all have conflicts with the own-ship, incidents involving the own-ship will much more likely to happen than those only involving pure intruders.

We need to run simulations of encounters and monitor the results of the simulations to decide whether they will lead to incidents or not. As a result, the multi-objective search problem cannot be fully represented in mathematical formulations, thus cannot be solved by classical mathematical programming techniques, such as Newton's method and its many relatives and variants. Consequently, we treat the simulation as a black-box and use a population-based evolutionary search method. Specifically, we combine agent-based *simulation* and a multi-objective *search* (specifically, Non-Dominated Sorting Genetic Algorithm II (NSGA-II) [28]) to automatically evolve encounters and find encounters that satisfy the two objectives. The method is a testing framework as is schematically shown in Fig. 1.

[5] We note that some dangerous situations may well exist that don't start with intruders in conflict with own-ship (but where conflict resolution action then places it in conflict with them later). Similarly, we acknowledge that some intruder-intruder accidents might be interesting because they are caused by own-ship's actions. However, for our purposes in this paper it is reasonable to exclude these cases, but anyone practically applying this technique, and seeking to get maximum results from it, would be well-advised to explore them.

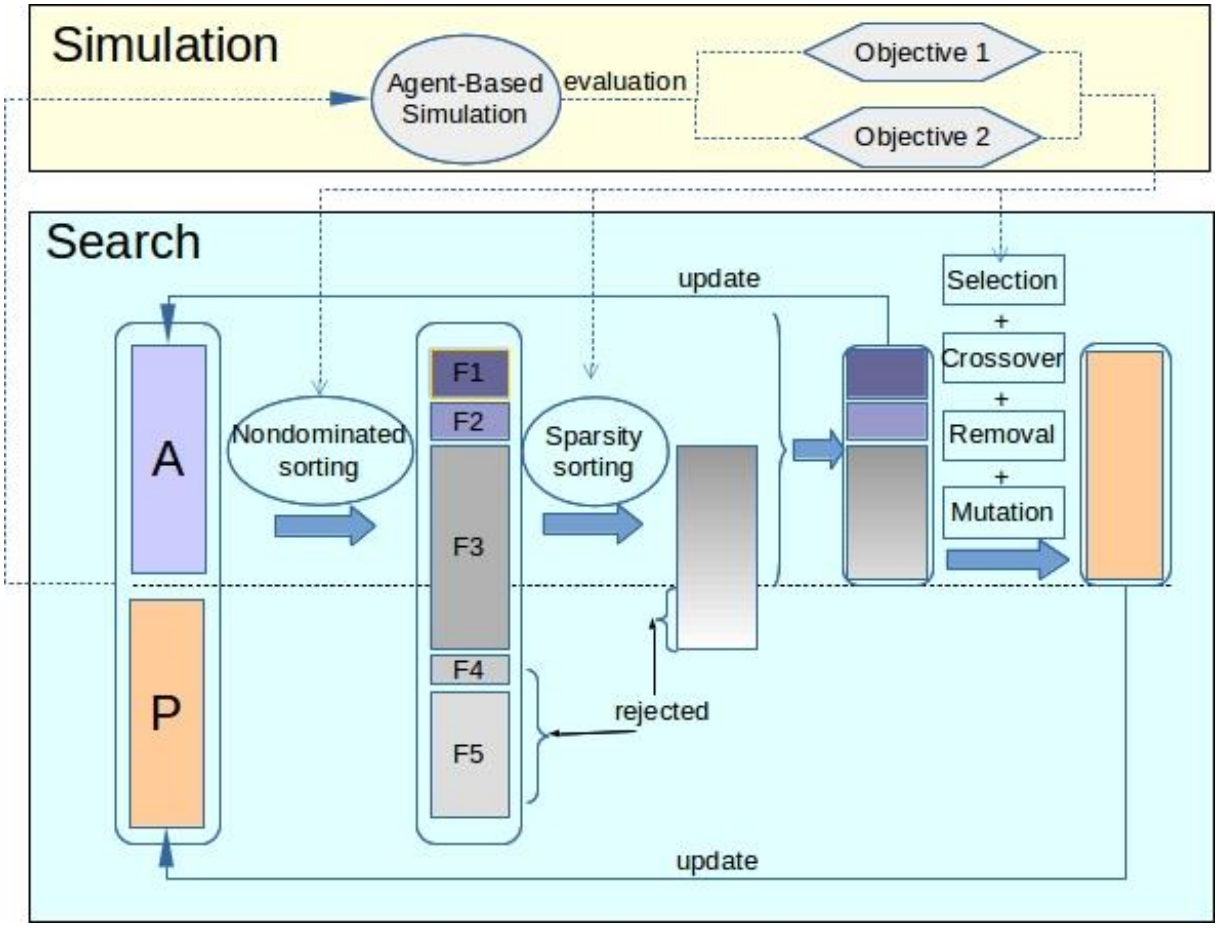


Fig. 1 Overview of the testing method that combines agent-based simulation and multi-objective search.

In this testing framework, encounters (candidate solutions) are treated as individuals that evolve by the law of "survival of the fittest", where fitness is determined based on how well they meet the two objectives. As noted above, simulation is treated as a black-box: its inputs are various encounters and its outputs are quantitative measurements of the simulated result of the encounters (fitness). **P** is the population holding the individuals of a generation, and it is initialized randomly. **A** is an archive to hold the "best" individuals ("elites") found through the history of the evolution, and is initialized to be empty. We have designed a representation for encounters in terms of the genome of individuals — each individual thus describes a (multi-intruder) encounter. Individuals in **P** and **A** are evaluated with respect to the two objectives using agent-based simulation. According to the evaluation result, individuals are ranked (using Non-dominated Sort and Sparsity Sort) and good individuals (those that have high rankings) are used to update the old elites in **A**. These good individuals also compete to be selected to breed the new population through genetic operators (genome cross-over, gene removal and gene mutation, see Section V). The

search process repeats until the termination condition (see Section V) becomes true. **F1** is the Pareto Front, which is a collection of the best individuals (more will be explained in Section V). The output of the process is the individuals (i.e. encounters that satisfy the two objectives) in **F1**.

We note that Alam et al. in [22, 23] and Clegg et al. in [24] used similar approaches for safety analysis of ATM systems. Their main concern is to identify the combination of airspace configurations and Air Traffic Controller actions that can result in a high aircraft collision risk. In contrast, in our work we are seeking specific counter-examples to challenge conflict resolution algorithms. While their work could be adapted for our purposes, no follow-up work in this direction can be found and the adaptation work may be considerable. And since their specific code is not publicly available, it is not possible to build directly on their work; for our work, the code will be open-source (for code, see links provided in Section VI) and further research can thus be easily built on it. Srikanthakumar et al. in [25] developed an automatic approach to finding worst cases for moving obstacle avoidance algorithms based on simulations and optimization techniques (one of which is a GA). However, their optimization problem has only one objective — to find a worst case situation that causes a collision. As a result, the worst case situations found may happen so rarely in real world conditions that they are not actually very helpful in improving the obstacle avoidance algorithms.

In the following three sections, we describe the design of our agent-based simulation, the auto-generation of encounters, and the detailed multi-objective search process.

III. Agent-Based Simulations

We use agent-based simulations to evaluate different kinds of encounters with respect to the two objectives identified above. Agent-based simulation [26] is selected for two main reasons: (1) it naturally models the multi-body interaction problem; and (2) it is already widely used in air traffic simulations.

We use MASON[6] as our simulation framework. We chose MASON mainly because it is open source and the user can easily control the fidelity of the simulation so that it can be run faster than real-time. In a typical agent-based simulation, there are three basic elements: agents, the environment, and their interactions. These elements are described in the following three subsections. The core of the simulation engine is a component known as the "Scheduler". All the agents are registered with this Scheduler, but with different scheduling frequencies and

[6] http://cs.gmu.edu/~eclab/projects/mason/

priorities. The simulation proceeds by scheduling the corresponding registered agents to conduct their defined behaviors at each simulation step.

A. Agents

There are two kinds of agent in our simulation: UAVs and global monitoring agents. UAVs have attributes, such as position, velocity, size and performance attributes etc. Some of the important performance limitations for the UAVs are listed in Table 1, which are based on those of the Parrot AR.Drone and the DJI Phantom UAVs. UAVs also have behaviors, such as flying to their targets, sensing other UAVs and avoiding conflicts with them. The conflict resolution algorithms are incorporated into the UAVs.

Global monitoring agents include a "Proximity Measurer" and an "Incident Detector". The Proximity Measurer measures the proximities (in horizontal distance and vertical distance) between the own-ship and the intruders at each simulation step, and records the minimum proximity experienced by the own-ship so far in a simulation. The Incident Detector monitors the simulations and detects any incidents involving the own-ship. However, the incidents between intruders are not monitored due to reasons mentioned in Section II.

Table 1 The UAV performance limits

Min Ground Speed	0m/s	Max Ground Speed	10m/s
Min Vertical Speed	-10m/s	Max Vertical Speed	2m/s

B. Environment

The environment in our simulation is a 3-D cuboid flight area (limited to $1000\text{m} \times 1000\text{m} \times 300\text{m}$ in length, width and height respectively). The horizontal area is arbitrary, but adequate for the analysis we are carrying out given the limited speeds of the UAVs. The vertical limit is based on some proposed regulations for commercial UAVs, e.g. those in [27] — we assume they are only permitted to fly below *300* meters in the airspace.

Apart from the agents described above, some other entities in the environment are the starting point and target (end point) for each UAV. In a simulation, UAVs fly from their starting points to the corresponding targets. If, due to conflict resolution, a UAV flies out of this cuboid flight area, it will not be monitored by the global monitoring agents any more. We have not modeled the effect of wind and other (static) objects, such as buildings, but this may be one of directions for future study.

C. Interactions

The interactions between the UAVs are only via the conflict resolution algorithms. We assume in each simulation step that the UAVs broadcast their state information (position, velocity and size) via ADS-B[7] or its equivalent. We explicitly model the sensor noise by adding some kinds of noise (e.g. white noise) to the received information by each UAV. We have not modeled any other communication between UAVs. The interactions between UAVs and the environment include UAVs following way points and generating new way points according to the conflict resolution algorithm.

IV. Automated Encounter Generation

We have devised a parameterized representation of pairwise encounters, which can be used to generate an intruder with respect to the own-ship by passing in the corresponding arguments. Multi-UAV encounters are generated by combining different intruders with the own-ship, which has a fixed initial state.

A. Parameterized Pairwise Encounters

A pairwise encounter involves two UAVs, one of which is the own-ship and the other is an intruder to be generated to conflict with the own-ship. To describe the state of a UAV for the conflict resolution problem, we use three parameters — position, velocity, and size. We model the UAVs as spheres with radii set according to the distances for safe separation — 10m in our case. The velocity can be represented either by three velocity components in each dimension as $[Vx, Vy, Vz]^T$, or by ground speed, bearing and vertical speed as $[Gs, Bearing, Vs]^T$. This is illustrated in Fig. 2(a) and the relationship between these two representations is as equation (1):

$$\begin{bmatrix} Vx \\ Vy \\ Vz \end{bmatrix} = \begin{bmatrix} Gs * \cos(Bearing) \\ Vs \\ Gs * \sin(Bearing) \end{bmatrix} \quad (1)$$

[7] ADS-B (Automatic Dependent Surveillance-Broadcast) is a cooperative surveillance technology with which a UAV will send its real time information, such as position and velocity, to its peers via a radio frequency.

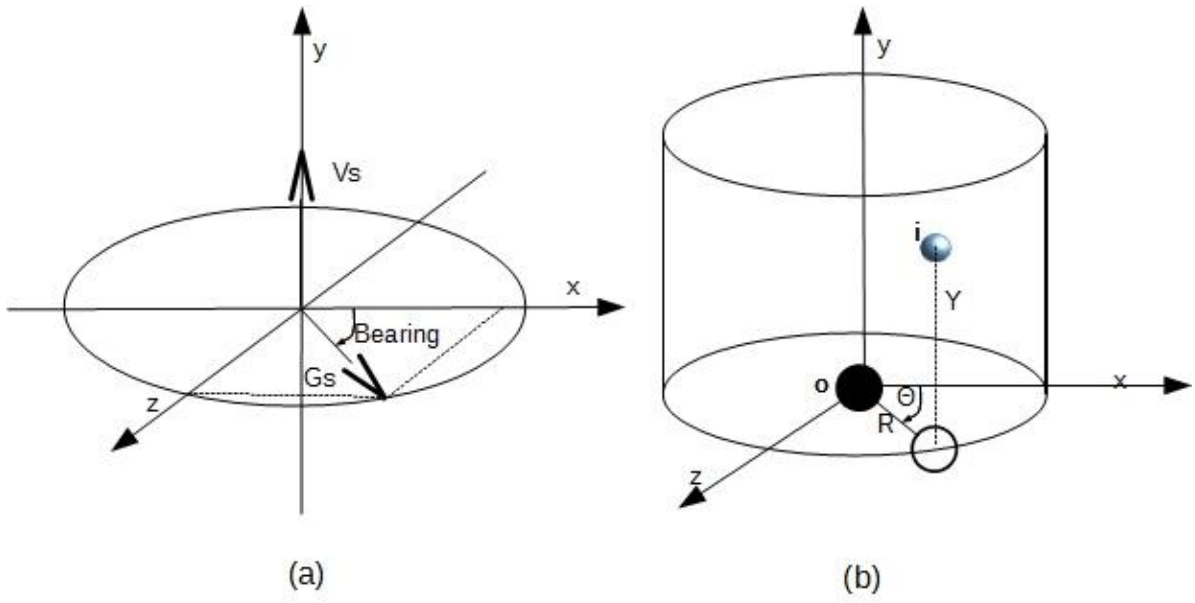


Fig. 2 (a) Two representations of the UAV's velocity. (b) Illustration of the relative position of the intruder (i) with respect to the own-ship (o) at the CPA. The own-ship is at the origin.

Assuming that the initial state of the own-ship has been fixed (in other words, we fix the initial position and velocity of the own-ship in a simulation), an intruder can be described by specifying the time for the own-ship and the intruder to arrive at the Closest Point of Approach (CPA), the intruder's relative position at the CPA with respect to the own-ship, as is shown in Fig. 2(b), and the intruder's velocity at the CPA. Seven parameters are used to specify the state of the intruder, which are:

- The time (T) left for the intruder to arrive at the CPA.
- The horizontal distance (R) between the two UAVs at the CPA, the angle (θ) of this approach, and the vertical distance (Y) at the CPA;
- The velocity $[Gs_i,\ Bearing_i,\ Vs_i]^T$ of the intruder at the CPA;

So, if the own-ship's initial position is $[X_o,\ Y_o,\ Z_o\ ]^T$ and its initial velocity is $[Gs_o,\ Bearing_o,\ Vs_o]^T$, the initial velocity and the initial position of the intruder can be defined by the vector equation (2) and (3):

$$\begin{bmatrix} Vx_i \\ Vy_i \\ Vz_i \end{bmatrix} = \begin{bmatrix} Gs_i * \cos(Bearing_i) \\ Vs_i \\ Gs_i * \sin(Bearing_i) \end{bmatrix} \tag{2}$$

$$\begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix} = \begin{bmatrix} X_o \\ Y_o \\ Z_o \end{bmatrix} + \begin{bmatrix} Gs_o * \cos(Bearing_o) \\ Vs_o \\ Gs_o * \sin(Bearing_o) \end{bmatrix} * T + \begin{bmatrix} R * \cos(\theta) \\ Y \\ R * \sin(\theta) \end{bmatrix} - \begin{bmatrix} Vx_i \\ Vy_i \\ Vz_i \end{bmatrix} * T \tag{3}$$

Varying the values of the seven parameters can generate different intruders, all posing a threat to the own-ship. In our simulation, we developed a pairwise encounter generator. If properly limiting the ranges of these parameters according to the performances of the simulated UAVs (see Section VI.B for specific values), a random pairwise encounter can be generated by uniformly selecting the values for the parameters from their ranges.

B. Multi-UAV Encounters

Multi-UAV encounters are generated by first fixing the initial state of the own-ship, and then generating various intruders using the parameterized pairwise encounter representation. In our case, the initial position of the own-ship is fixed at the central left of the simulated flight space for better visualization. Its initial velocity vector is specified by a normal ground speed (*5m/s*), a bearing directly to the right (*0°*) and a zero vertical speed. Its size is specified by a sphere with radius of *10m*. It is noted that many conflict resolution algorithms are based on relative geometry positions and velocities, so that fixing the initial state of the own-ship usually will not make much harm to the simulation. After the initial state of the own-ship is fixed, various intruders can be generated by passing the specific arguments to the pairwise encounter generator. The generated intruders are combined with the own-ship to form a multi-UAV encounter. For example, if we pass the two groups of parameters shown in Table 2 to the encounter generator, a multi-UAV encounter will be generated and the simulation of it (with conflict resolution algorithms functioning) is shown in Fig. 3.

Table 2 The parameters for the generation of an example multi-UAV encounter

T	R	θ	Y	Gs_i	$Bearing_i$	Vs_i
20	5	90	17	7	131	-2
18	0	90	8	10	0	1

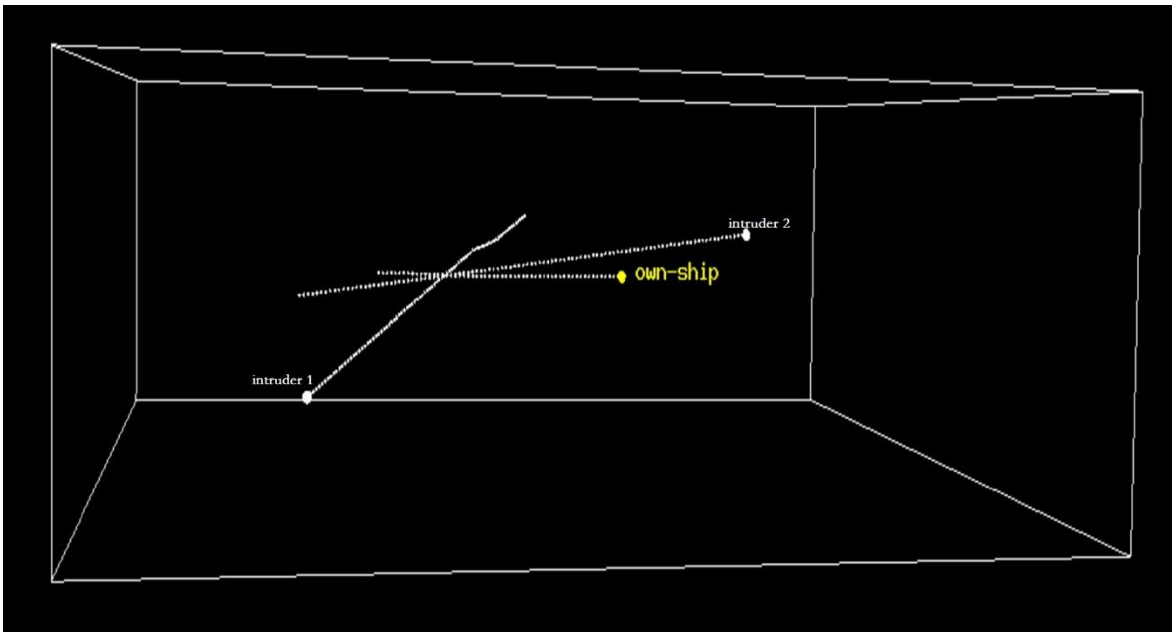


Fig. 3 The simulation of a multi-UAV encounter generated by the parameters in Table 2.

The first row of arguments specifies a left-crossing intruder (intruder 1) (with a crossing angle of 131^o), and the second row of arguments specifies an overtaking intruder (intruder 2) since its bearing is 0^o and its ground speed is faster than the own-ship's ($10m/s > 5m/s$).

V. Genetic Algorithm Based Multi-Objective Search

GAs are population-based evolutionary search methods. By "population-based", we mean that the algorithm holds a set of candidate solutions to a specific problem. By "evolutionary", we mean that these candidate solutions evolve by a mechanism mimicking natural evolution ("survival of the fittest"). In each generation of the evolution, the population retains the solutions most likely to solve the problem and the others are eliminated.

For the multi-objective search problem formulated in Section II, we solve it based on the Non-Dominated Sorting Genetic Algorithm II (NSGA-II) [28], which is a specific form of GA. Referring to Fig. 1, the NSGA-II procedure is as follows.

1. Set up a population (**P**) to hold encounters as individuals in a generation. The size of **P** is n and the initial individuals are randomly generated;
2. Set up an Archive (**A**) to hold the best individuals (elites) found through the history of the evolution. The size of **A** is also n and **A** is initialized to be empty;

3. Run simulations to evaluate individuals in **P** and **A** with respect to the two objectives;
4. Rank the individuals in **P** and **A** according to the evaluation result. Store the best individuals in a collection **F1** (**F1** for "rank 1 Pareto Front" — see Section V.C, below, for an explanation);
5. Select from the individuals the best *n* individuals to update the old elites in **A**;
6. Run tournament selection with replacement to select *n* individuals from the new elites in **A**;
7. Breed a new population from the selected individuals through genetic operators (genome cross-over, gene removal and gene mutation) and update the old population **P**;
8. When (1) an ideal individual is found[8], or (2) the allotted time is over; or (3) a certain number of generations have been evaluated, terminate the process and output the individuals in **F1**. Otherwise, repeat steps 3-8.

We implemented this evolutionary multi-objective search by using ECJ[9], which is a Java-based evolutionary computation system. In the following subsections, we detail the design of the search technique.

A. Encoding Encounters with Genomes

A multi-UAV encounter is a candidate solution to our multi-objective search problem, and it is represented in the search as an *individual*. Each individual has one and only one *genome*, which is a variable-length collection of *genes*. Each gene has seven slots to store the seven arguments for generating an intruder. The genome of an individual thus encodes a multi-UAV encounter. The evolution of the individuals is thus the improvement of the multi-UAV encounters so that they (are more likely to) satisfy the two objectives. This relationship is shown is Fig. 4.

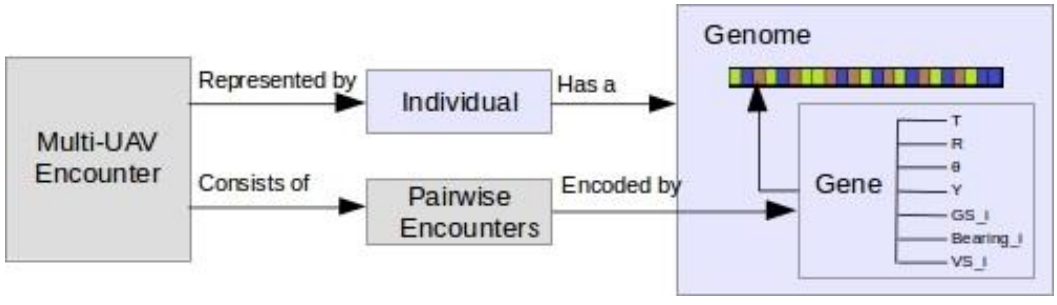


Fig. 4 Encoding multi-UAV encounters in genomes for the use of the GA based multi-objective search.

[8] An ideal individual is unlikely to be found in our case due to the opposing nature of the two objectives.
[9] http://cs.gmu.edu/~eclab/projects/ecj/.

B. Objectives

To quantify the values of the two objectives in our problem (see Section II), we define them formally as:

$$Objective\ 1 = \frac{1.0}{1 + p_{min}} \tag{4}$$

$$Objective\ 2 = 1.0 - \frac{|\mathrm{E}| - 2}{C_{max} - 2} \tag{5}$$

In equation (4), p_{min} is the scalar value of the minimum proximity experienced by the own-ship with any intruders in a simulation run, which is defined as equation (6):

$$p_{min} = min_{s\epsilon[0,S]}\{min_{k\epsilon[1,|E|-1]}\{p_{k_s}\}\} \tag{6}$$

where S is the number of the total simulation steps, |E| is the cardinality of an encounter E, p_{k_s} is the scalar value of the proximity between the own-ship and the k^{th} intruder in the simulation step s; the value of proximity between two UAVs is defined by equation (7):

$$p = max\{0, distH - H\} + max\{0, distV - V\} \tag{7}$$

Where $distH$ and $distV$ are respectively the horizontal and vertical distance between the center points of the two UAVs, H and V are the required horizontal and vertical distance of safe separation (both are *20m* in our case).

In equation (5), C_{max} is the maximum cardinality allowed in the search problem, in our case it is *11* (i.e. at most *10* intruders are allowed in an encounter[10]. The *-2* is because there should always be at least two UAVs — the own-ship and at least one intruder.

The definition of the objectives is such that larger values are better. For objective 1, if there is an incident, p_{min} will be *0*, and it will get its largest value, *1.0*. For objective 2, if there is only one intruder (the cardinality is *2*), its value is *1.0*, which is the largest possible value.

C. Selection of the Fittest Individuals

In multi-objective search, it is often the case that there is no optimal solution that achieves all objectives simultaneously. Instead, there is a set of "best options" which are equally good. To find these best options, we need to define what it means for one solution to be "better" than another. Suppose there are only two candidate solutions, S_1 and S_2. S_1 is said to be better than S_2 if and only is S_1 is *at least as good as* S_2 in all objectives and is better than S_2 in at least one objective. If this is the case, S_1 is said to *Pareto dominate* S_2. Neither S_1 nor S_2 Pareto dominates the

[10] This number was used for the work described here, but it can be changed to a bigger number if the search cannot find any encounters satisfying the two objectives in reasonable time.

other if they are equally good in all objectives, or if S_1 is better in some objectives while S_2 is better in others. In those cases, both S_1 and S_2 are best options and we say they are on the *Pareto Front* of the space of candidate solutions. The main target of our multi-objective search is to find those solutions at the Pareto Front.

For a population of individuals (candidate solutions), we can compute the Pareto Front in the following way: go through the population and add an individual to the front if it is not Pareto dominated by any individual currently in the Pareto Front, and remove individuals from the front if they are dominated by this new individual [29].

The individuals in the population can be grouped according to how close an individual is to the Pareto Front. To do this, we assign a rank to each individual. Individuals presently in the Pareto Front are in rank 1. If we removed the rank 1 individuals from the population and compute the new Pareto Front, the individuals in the new front are assigned a rank of 2. Likewise for rank 3, and so on until no individuals remain. This is the mechanism for the *Non-dominated Sort*, which is used by the NSGA-II as the prime criteria to rank individuals. Individuals with rank *r* are stored in $\mathbf{F_r}$ and individuals with lower ranks are first selected as elites to form the Archive (**A**). For full details, readers are referred to [29].

Considering the second criteria, if there are too many individuals with the same rank competing to be selected as elites (since the size of the **A** is fixed at *n)*, those evenly spread across that front would be chosen. The NSGA-II achieves this purpose by defining the *sparsity* for an individual. An individual is of high sparsity if the closest individuals on either side of it on the Pareto Front aren't too close to it. For the definition and computation of sparsity and the *Sparsity Sort*, readers are referred to [29].

After the new elites are selected, they compete to be selected as the fittest individuals to breed the new population. The process is a *Tournament Selection* [29], where individuals are randomly selected with replacement to compete with the champion so far. The criteria for competition are first based on rank and then on sparsity.

D. Genetic Operators

After selecting the fittest individuals, their genomes are modified randomly ("tweaked") in the hope of generating better individuals. As with biological evolution, not all such operations are beneficial, but some are, and those are the ones that will most likely survive into later generations (see sub-section E). We use three genetic operators to tweak the genome: genome cross-over, gene removal, and gene mutation. In the evolutionary process, all the genetic operators are called to operate on individuals with certain probabilities in each generation.

Genome cross-over involves mixing and matching parts of two old genomes to form new genomes. We use the one-point cross-over approach, which randomly selects a position for cross-over and swaps the genes after that position, as is illustrated in Fig. 5.

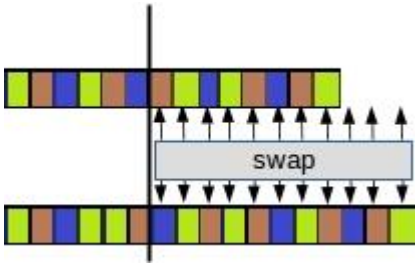


Fig. 5 Illustration of the one-point genome cross-over operator.

Gene removal selects one gene in the genome (with uniform probability of each) and removes it. If the length of the genome is only one gene, then it does nothing. Gene removal is used to reduce the number of intruders involved in an encounter, thus favoring the second objective.

Gene mutation has a certain probability (*prob*) of mutating each gene in a genome. If a gene is selected to mutate, Gaussian noise ($\sim N(0,\ \sigma^2)$) is applied to the information stored in that gene. For example, in our case, there are 7 genes in a genome, and for each gene, the random number generator generates a random real number from range [0, 1]. If this number is less than *prob*, this gene will be mutated. Given that the value stored in this gene is *value*, then the new value ($value'$) in the mutated gene is defined by equations (8) and (9).

$$value' = (1 + rnd) * value \tag{8}$$

$$rnd \sim N(0,\ \sigma^2) \tag{9}$$

E. Formation of the Next Generation

In the most common form of GA, the next generation of the population is generated from the old population. NSGA-II differs from this — it holds an archive of the same size as the population which contains the best *n* individuals (elites) found so far. Every generation, the population competes with the existing elites to be selected as one of the new elites. After the new elites are selected, they are used to breed the next generation of the population.

VI. Case Study: ORCA-3D

A. A Brief Explanation of ORCA-3D

ORCA-3D (Optimal Reciprocal Collision Avoidance in 3D) [30] is a cooperative collision avoidance algorithm for multi-agent systems. The ORCA algorithm is an improvement of the Collision Cone approach [31], specifically, it avoids the oscillations often experienced in collision cone applications. Here, we are using ORCA for UAV conflict resolution — to do this, we enlarge the collision distance to the safe separation distance. In the remainder of this subsection, we will give an explanation of how ORCA-3D works. Readers who are not interested in that can safely skip to subsection B.

As its name suggests, ORCA-3D works in 3D space, but to simplify our explanation here we will use only two dimensions. We assume **A** and **B** are two agents moving in the 2-D plane. Let $\mathbf{P_A}$, $\mathbf{V_A}$ and $\mathbf{r_A}$ denote the current position, velocity vector and radius of agent **A** respectively, and let $\mathbf{P_B}$, $\mathbf{V_B}$ and $\mathbf{r_B}$ be the position, velocity vector and radius of agent **B** respectively, as is shown in Fig. 6(a). We define the *Collision Cone*, which is written as $\mathbf{CC_{A|B}}$, as the set of colliding *relative velocities* ($\mathbf{V_{rel}}$) between **A** and **B**. If **A** and **B** maintain a relative velocity in $\mathbf{CC_{A|B}}$, a collision will happen at some future moment, say t. Formally, $\mathbf{CC_{A|B}}$ is defined by equation (10):

$$\mathbf{CC}_{A|B} = \{\boldsymbol{V}_{rel} | \exists t > 0 : \boldsymbol{V}_{rel} * t \in D(\boldsymbol{P}_B - \boldsymbol{P}_A, \boldsymbol{r}_A + \boldsymbol{r}_B)\} \tag{10}$$

where the notation $D(\boldsymbol{x}, \boldsymbol{r})$ represents a disc with center $\boldsymbol{x}$ and radius $\boldsymbol{r}$. A geometric interpretation of $\mathbf{CC}_{A|B}$ is shown in Fig. 6(b).

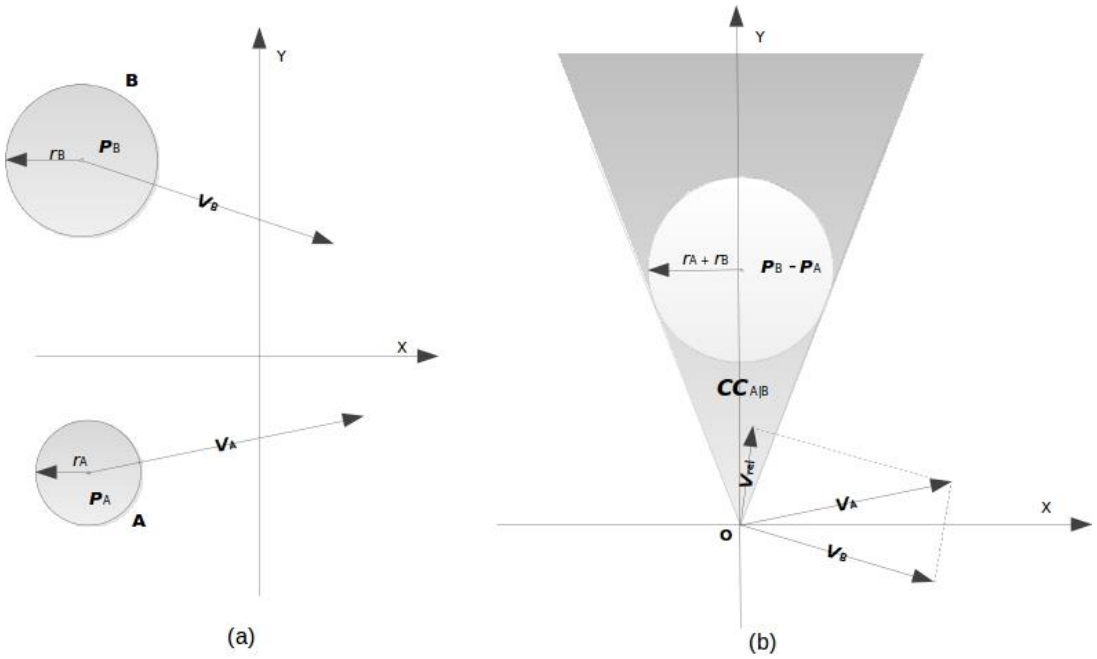


Fig. 6 (a) A configuration of two agents in a 2D plane. (b) The Collision Cone $\mathbf{CC}_{A|B}$. If the relative velocity $\boldsymbol{V}_{rel}$ is inside $\mathbf{CC}_{A|B}$, a collision will happen at some future moment.

It follows that if the two agents choose a relative velocity outside $\mathbf{CC}_{A|B}$, then a collision will not occur in a time horizon t. For a decentralized system, this can be achieved in one of two approaches: (1) one of the agents chooses a new velocity vector while assuming the other will maintain its old velocity, or (2) the two agents cooperatively choose new velocities. In either approach, the new relative velocity should not be inside $\mathrm{CC}_{A|B}$. The original Collision Cone approach uses the first approach and it sometimes results in oscillations [32].

The ORCA approach uses approach (2) — given a minimum change for the relative velocity to be outside of $\mathbf{CC}_{A|B}$ denoted by $\boldsymbol{u}$, each agent is required to take at least half of the responsibility to make this change happen. So, agent **A** should change its velocity by at least $\mathbf{0.5}\boldsymbol{u}$ such that the end point of its velocity vector should fall in the half plane divided by a line ($\mathbf{L}_1$) through $\mathbf{V}_A + \mathbf{0.5}\boldsymbol{u}$ perpendicular to $\boldsymbol{u}$, as is shown in Fig. 7(a).

This provides one constraint for **A** to choose its new velocity.

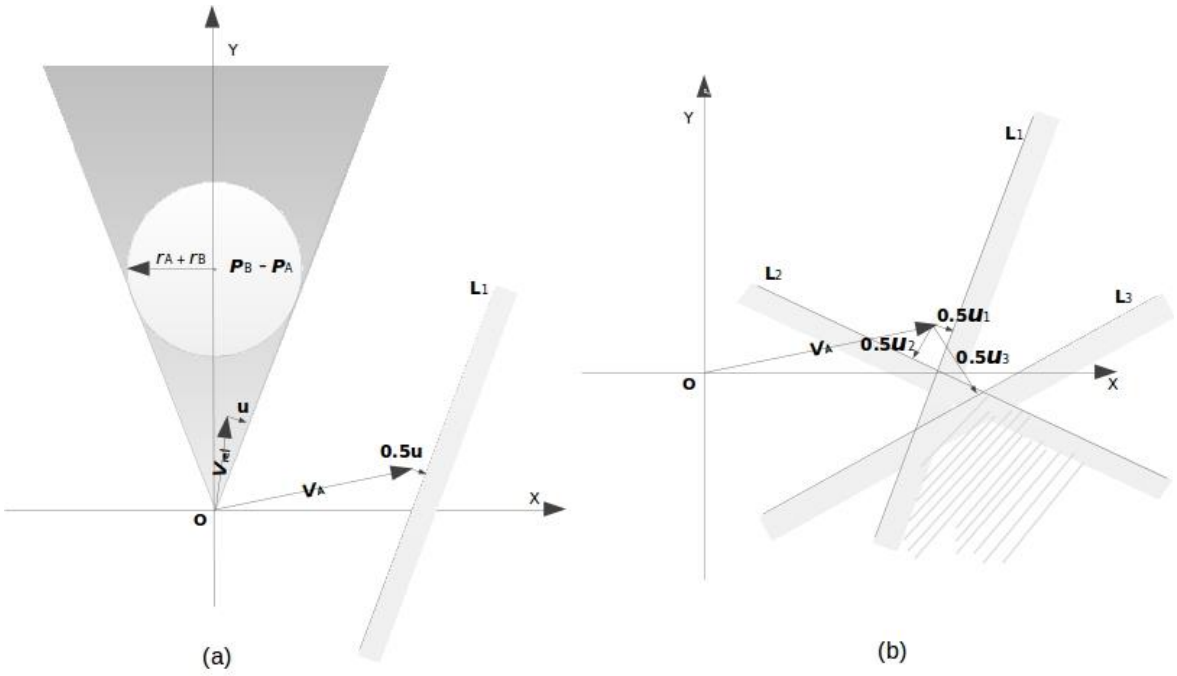


Fig. 7 (a) two-agent ORCA. (b) Multi-agent ORCA.

If there is more than one agent for **A** to avoid, each agent will exert a constraint for **A**, and the new velocity **A** chooses should satisfy all the constraints. The new velocity can be computed efficiently using linear programming, since the end point of the new velocity vector should fall inside the convex region surrounded by the lines ($\mathbf{L}_1$, $\mathbf{L}_2$, $\mathbf{L}_3$ ...) denoting the half-planes. See Fig. 7(b) for an illustration.

If we extend the above algorithm to three dimensions, we have ORCA-3D (the use of the term "cone" above illustrates how the algorithm is extended).

For an agent to use the ORCA-3D approach, the only information it needs about the other agents is their current relative positions, relative velocities and shapes. Here we assume that each UAV is fitted with ADS-B or its equivalent to broadcast the information to its peer UAVs. We also assume the noise to this information follows normal distributions. Further failures of the ADS-B are not modeled.

There is an open source C++ implementation of ORCA-3D[11], but we re-implemented[12] it in Java for easy integration with the other parts of our framework. In the following two subsections, we report the use of a random search based approach and our proposed method to test the capability of the ORCA-3D algorithm for multi-UAV conflict resolution. All the experiments[13] were done on a PC with an Intel Core i3-2350M 2.30GHz CPU and the 64 bit Ubuntu 14.04 Operating System. The experiments were run using Java 7 with a maximum memory of 1024MB for the JVM.

B. Applying a Random Search Approach

In the random search method, we generate a stream of encounters using a simple random approach, terminating only when a certain (large) number of encounters have been generated and evaluated, or a certain (more modest) number of encounters that lead to incidents have been found. Assuming this process has indeed generated some encounters that lead to incidents (thus meeting the first objective that we gave in Section II), we then select a subset of those that have the lowest cardinalities (thus meeting the second objective).

For the first step, we use a process that repeats a simple step a large number of times — we randomly generate a multi-UAV encounter, then run a simulation to decide whether or not it can lead to an incident. If an incident happens, this encounter is recorded. As noted above, this process terminates when a certain (large) number of encounters have been generated and evaluated, or enough encounters that lead to incidents have been found. In this case, we set limits of *100,000* encounters overall and *500* incidents. The cardinality of the generated encounters is random, between *2* to *11* — the upper limit was set at a value we thought should be sufficient to "stress" the conflict

[11] Original ORCA-3D can be found in http://gamma.cs.unc.edu/RVO2/downloads/.
[12] Our Java implementation of ORCA-3D can be found in https://github.com/superxueyizou/ORCA_3D_UAV.git.
[13] The code for the experiment can be found and downloaded from https://github.com/superxueyizou/ORCA-3D-Testing.git.

resolution algorithm. The bounding values for the parameters of the individual generated intruders are shown in Table 3 (see Section IV.A for the meaning of these parameters).

Table 3 Bounding values for the parameters

parameters	T	R	θ	Y	Gs	Bearing	Vs
min	10	0	-180	-20	2	-180	-2
max	30	20	180	20	10	180	2

A randomly generated encounter and the associated simulation run is visualized in Fig. 8. There are 8 intruders. In this case, with the help of the ORCA-3D conflict resolution, every UAV reached its target safely.

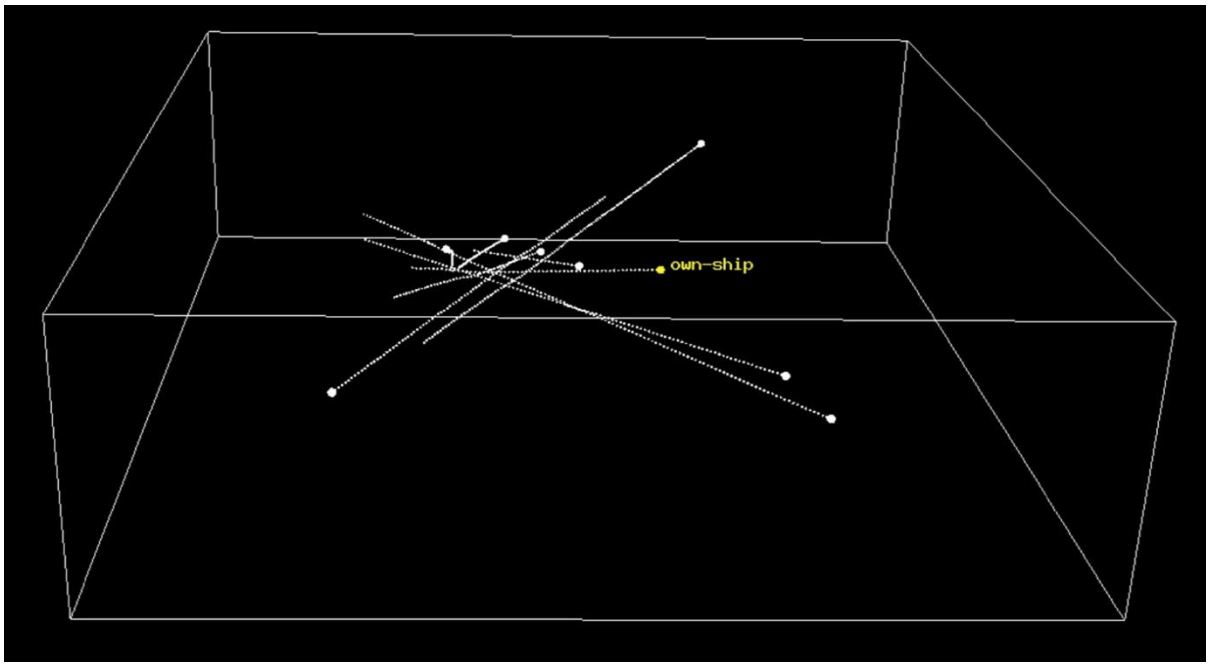


Fig. 8 A simulated random encounter of 9 UAVs. No incident occurred.

We conducted *5* trials, each with a different number as the seed for the random number generator. For each trial, we generated and evaluated *100,000* encounters. So, in all we ran simulations for *500,000* encounters. However, no incidents occurred. The time it took for each trial is shown in Table 4. It can be noted that these times are not huge — even using a single ordinary PC.

Table 4 Time cost for each trail

seed	97846789	194679667	249719121	567971664	946163716
time	412s	410s	410s	397s	395s

Since we cannot find enough encounters that can lead to an incident (indeed, we found none at all), we cannot proceed with the second step. The random search based approach failed, at least in this case. Since no incidents occurred in these *500,000* encounters, it appears that the ORCA-3D algorithm is very likely to be capable of handling more than *11* UAVs.

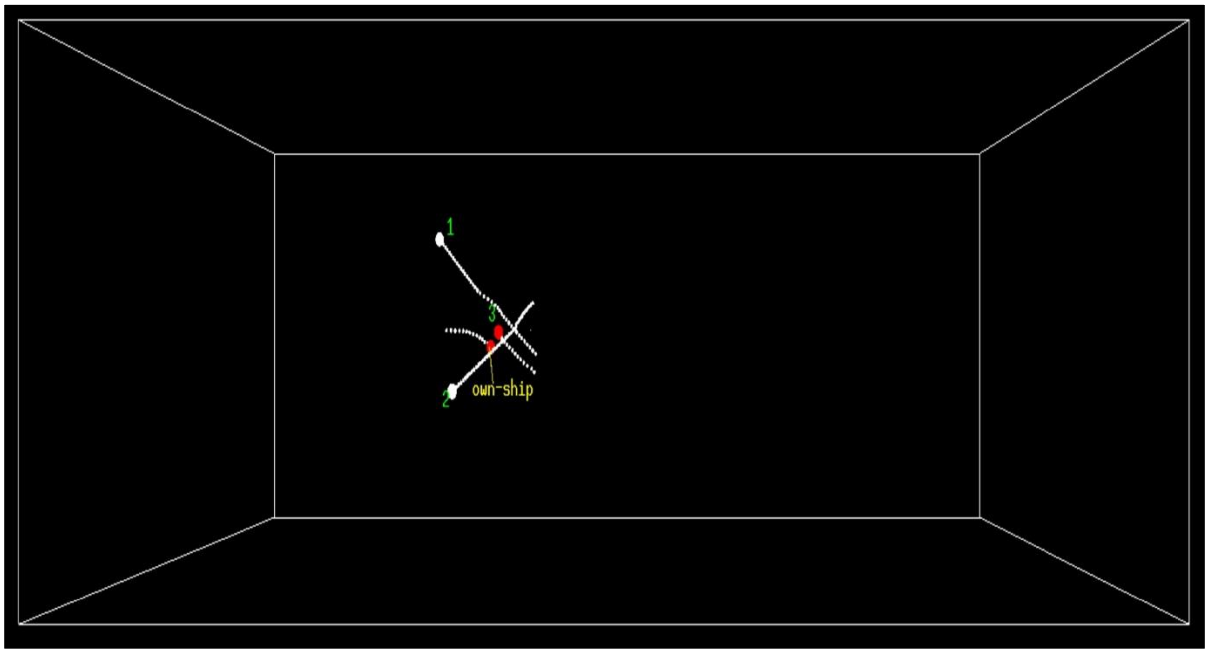
C. Applying Our Proposed Evolutionary Search Method

We also used our proposed method to test the capability of ORCA-3D. The NSGA-II algorithm was set to evolve for *20* generations, each generation having a population of *5,000*. So, in this case, we also evaluated *100,000* encounters in a search. For the initial population, the length of the genome is uniformly chosen from *1* to *10*, inclusive, giving a maximum cardinality of encounters of *11* as before. The bounding values of the parameters are the same as those used in the random search approach. Again we ran *5* searches with different seeds for the random number generator. The result is shown in Table 5. In this table, row 1 lists the seed for each search; row 2 lists the time cost for each search; row 3 lists the number of encounters in the Pareto Front; row 4 lists the number of encounters in the Pareto Front that can lead to an incident; and row 5 lists the minimum number of intruders in an encounter that can lead to an incident.

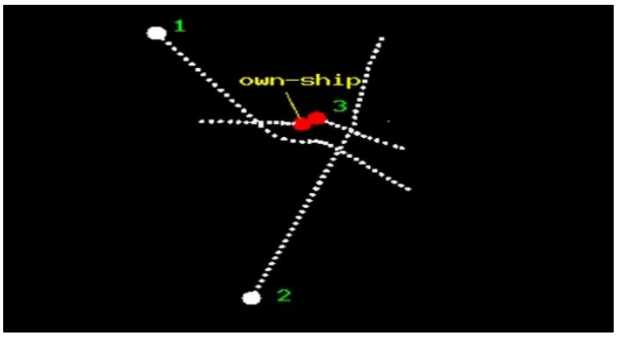
Table 5 Statistics of the 5 searches using our proposed method

seed	567672542	588764257	679463479	884185791	898946497
time	221s	213s	226s	169s	198s
in Pareto Front	27	18	6	49	13
with incidents	20	11	0	0	8
fewest intruders	9	**3**	>10	>10	9

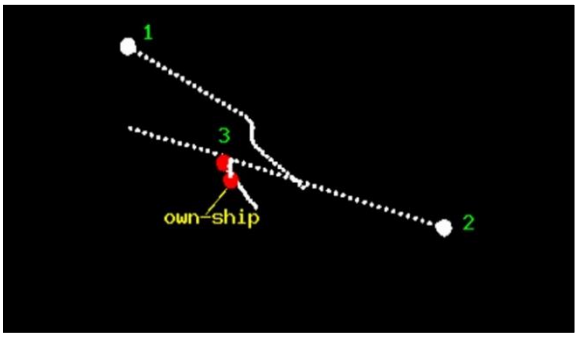
In these multi-objective searches, we are interested in encounters in the Pareto Front. Three out of five searches found encounters that can lead to incidents in the Pareto Front. The minimum number of intruders that can cause an incident with the own-ship is *3*. This 4 UAV encounter is shown in Fig. 9.



(a) Front view



(b) Top view



(c) Side view

Fig. 9 A simulated 4 UAV encounter that leads to an incident.

In this encounter, the own-ship first traveled at a constant level from left to right. At the time when the maneuvers began, intruder 1 was climbing in above the own-ship and intruder 2 was flying outwards from the left of the own-ship and also above the own-ship. This caused the own-ship to descend. But intruder 3 was climbing from below the own-ship, and a violation of safe separation happened.

Since we had found an encounter with a cardinality of *4* that can lead to an incident, we conclude that the ORCA-3D algorithm cannot handle *4* UAVs in all cases. The exact reason for the incident could be determined with further analysis of this encounter, which is beyond the scope of this paper. For our purposes is it sufficient to observe that with certain traffic patterns ORCA-3D is not able to find paths that avoid the separation violation with the own-ship, because of the constraints imposed by the other UAVs.

D. Discussion

Our proposed method can find incidents which the random search approach cannot easily find. The key difference between the two is the former's utilization of metaheuristic search using NSGA-II. NSGA-II's key strength is that it maintains an archive of best candidate solutions (elites) found so far, and breeds further candidate solutions from those best candidates. Each new generation of the population therefore tends to have desirable features descended from these elites. After each generation, the addition of the best new individuals to the set of elites general leads to the set of elites improving over time.

In order to have the new generation inherit the good features of the elites, the desirable features should be quantified and embodied in the fitness function (the quantification of the objectives). In our case, the good features we choose are (1) a lower proximity between the own-ship and the intruders in a simulated encounter, and (2) a lower cardinality of the encounter. They are quantified and embodied in the valuation functions of the two objectives. Better selection of features[14] may, in the future, make our method even more powerful.

Our proposed method can find the encounters meeting the two criteria more quickly (in a practical time with modest processing facilities) than our random search approach. The random search based approach takes about *400s* on average to explore *100,000* encounters, while our approach only takes about *210s*. For the random search based approach, the main time expense is the simulation runs to evaluate these encounters, while for our method, the expense includes two parts: the simulation runs and the overhead of the multi-objective search framework. Our method is faster because the evolutionary search algorithm favors individuals that score well on our low-cardinality objective, and thus the simulated encounters tend to have fewer UAVs involved. This reduces the computing effort required to simulate them.

Our proposed method is probably most valuable in the early stage of UAV conflict resolution algorithm development. It can quickly find challenging encounters for an algorithm and can help to find its limitations, such

[14] The selection of good features is, as ever, problem-specific, and relies on a deep understanding of the problem.

that the algorithm can be improved. One weakness of our method is that there is no way to assign statistical confidence to the results — it is effective at fault-finding, but not at providing confirmatory evidence of fault-freeness. In contrast, random search approaches (when their probabilities are tuned to match actual patterns of operational data) can provide such confidence — see, for example, the work of Stroeve et al as discussed in [33]. In practice, the two techniques may thus prove complimentary.

VII. Summary and Conclusions

In this paper, we tackled the problem of testing the capability of multi-UAV conflict resolution algorithms. The random search approach to this problem has some drawbacks, so we reformulated the problem as a multi-objective search problem. We developed a method to automatically find encounters meeting the objectives using agent-based simulation and multi-objective search. We devised a way to automatically generate encounters with any number of UAVs involved. Fast-time agent-based simulations were used to evaluate the encounters with respect to the objectives and a GA based multi-objective search method was designed to search for encounters satisfying the objectives. Experiments were conducted to compare our proposed method and the random search based approach by using the two to test the capability of ORCA-3D for conflict resolution.

The results show that the proposed method can find encounters meeting these objectives more efficiently than the random search based approach. The resulting encounters providing the starting points for further analysis of the conflict resolution algorithm, which will in turn allow algorithm developers and users to fully understand its limitations. Thus in general, our approach has the potential to offer an effective means of validating, or at least determining some limits of, SAA algorithms, thereby helping developers and regulators decide whether these important algorithms can be deployed

There are some limitations of our approach: (1) as mentioned in Section II, in our simulations, all intruders are generated to have conflict with the own-ship. However, we note that some dangerous situations may well exist that don’t start with intruders in conflict with own-ship (but where conflict resolution action then places it in conflict with them later); (2) we also ignore incidents between pure intruders and we are only interested in incidents involving the own-ship. We acknowledge that some intruder-intruder accidents might be interesting because they are caused by own-ship's actions. By doing so, some specific types of faults in the algorithm may be ignored; (3)

considering the small-size and operational scenarios of the studied UAVs, follow-up research may be needed to model the effects of wind and static obstacles in simulations.

Acknowledgements

Xueyi Zou would like to thank the China Scholarship Council (CSC) for its partial financial support for his PhD study.

References

1. FAA. "Integration of Civil Unmanned Aircraft Systems (UAS) in the National Airspace System (NAS) Roadmap." Federal Aviation Administration, 2013.
2. Kochenderfer, M. J., and Chryssanthacopoulos, J. P. "Robust Airborne Collision Avoidance through Dynamic Programming." Massachusetts Institute of Technology, Lincoln Laboratory, 2011.
3. Ghosh, R., and Tomlin, C. "Maneuver design for multiple aircraft conflict resolution," *Proceedings of the 2000 American Control Conference*. Vol. 1, 2000, pp. 672-676 vol.1.
4. Bicchi, A., and Pallottino, L. "On optimal cooperative conflict resolution for air traffic management systems," *IEEE Transactions on Intelligent Transportation Systems.* Vol. 1, No. 4, 2000, pp. 221-231.
5. Khatib, O. "Real-time obstacle avoidance for manipulators and mobile robots," *The international journal of robotics research.* Vol. 5, No. 1, 1986, pp. 90-98.
6. Jenie, Y. I., Van Kampen, E.-J., de Visser, C. C., and Chu, Q.-P. "Selective velocity obstacle method for cooperative autonomous collision avoidance system for UAVs," *AIAA Guidance, Navigation, and Control (GNC) Conference, Boston, MA*. 2013.
7. Menon, P., Sweriduk, G., and Sridhar, B. "Optimal strategies for free-flight air traffic conflict resolution," *Journal of Guidance, Control, and Dynamics.* Vol. 22, No. 2, 1999, pp. 202-211.
8. Richards, A., and How, J. P. "Aircraft trajectory planning with collision avoidance using mixed integer linear programming," *Proceedings of the 2002 merican Control Conference*. Vol. 3, 2002, pp. 1936-1941 vol.3.
9. Paielli, R. A., Erzberger, H., Chiu, D., and Heere, K. R. "Tactical conflict alerting aid for air traffic controllers," *Journal of Guidance, Control, and Dynamics.* Vol. 32, No. 1, 2009, pp. 184-193.
10. Erzberger, H., Lauderdale, T. A., and Chu, Y.-C. "Automated conflict resolution, arrival management, and weather avoidance for air traffic management," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of aerospace engineering*. 2011, p. 0954410011417347.
11. Bushnell, D., Giannakopoulou, D., Mehlitz, P., Paielli, R., and Pasareanu, C. "Verification and validation of air traffic systems: Tactical separation assurance," *Proceedings of the 2009 IEEE Aerospace conference*. 2009, pp. 1-10.
12. Havelund, K., and Pressburger, T. "Model checking JAVA programs using JAVA PathFinder," *International Journal on Software Tools for Technology Transfer.* Vol. 2, No. 4, 2000, pp. 366-381.
13. Chilenski, J. J., and Miller, S. P. "Applicability of modified condition/decision coverage to software testing," *Software Engineering Journal.* Vol. 9, 1994, pp. 193-200(7).
14. Giannakopoulou, D., Howar, F., Isberner, M., Lauderdale, T., Rakamari\'c, Zvonimir, and Raman, V. "Taming Test Inputs for Separation Assurance," *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*. ACM, New York, NY, USA, 2014, pp. 373-384.
15. Paielli, R. A. "Automated Generation of Air Traffic Encounters for Testing Conflict-Resolution Software," *Journal of Aerospace Information Systems.* Vol. 10, No. 5, 2013, pp. 209-217.
16. Blum, D. M., Thipphavong, D., Rentas, T. L., He, Y., Wang, X., and Pate-Cornell, M. E. "Safety analysis of the advanced airspace concept using Monte Carlo simulation," *Proceedings of the AIAA Guidance, Navigation, and Control Conference*. 2010.
17. Thipphavong, D. "Accelerated Monte Carlo simulation for safety analysis of the advanced airspace concept," *Proceedings of the 10th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*. Vol. 3, 2010, p. 5.
18. Blom, H. A., and Bakker, G. "Safety Evaluation of Advanced Self-Separation Under Very High En Route Traffic Demand," *Journal of Aerospace Information Systems.* 2015, pp. 1-15.
19. Zou, X., Alexander, R., and McDermid, J. "Safety Validation of Sense and Avoid Algorithms Using Simulation and Evolutionary Search," *Computer Safety, Reliability, and Security*. Vol. 8666, Springer International Publishing, 2014, pp. 33-48.

20. Roland, W., Matthew, E., and Caroline, F. "Establishing a Risk-Based Separation Standard for Unmanned Aircraft Self Separation," *11th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*. American Institute of Aeronautics and Astronautics, 2011.
21. Goldberg, D., and Holland, J. "Genetic Algorithms and Machine Learning," *Machine Learning.* Vol. 3, No. 2-3, 1988, pp. 95-99.
22. Alam, S., Lokan, C., and Abbass, H. "What can make an airspace unsafe? characterizing collision risk using multi-objective optimization," *Proceedings of the 2012 IEEE Congress on Evolutionary Computation (CEC)*. 2012, pp. 1-8.
23. Alam, S., Lokan, C., Aldis, G., Barry, S., Butcher, R., and Abbass, H. "Systemic identification of airspace collision risk tipping points using an evolutionary multi-objective scenario-based methodology," *Transportation Research Part C: Emerging Technologies.* Vol. 35, No. 0, 2013, pp. 57 - 84.
24. Clegg, K., and Alexander, R. "The Discovery and Quantification of Risk in High Dimensional Search Spaces," *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*. ACM, New York, NY, USA, 2013, pp. 175-176.
25. Srikanthakumar, S., and Chen, W.-H. "Worst-case analysis of moving obstacle avoidance systems for unmanned vehicles," *Robotica.* Vol. 33, 2015, pp. 807-827.
26. Macal, C. M., and North, M. J. "Tutorial on Agent-based Modeling and Simulation," *Proceedings of the 37th Conference on Winter Simulation*. Winter Simulation Conference, 2005, pp. 2-15.
27. "Overview of Small UAS Notice of Proposed Rulemaking." Vol. 2015, http://www.faa.gov/regulations_policies/rulemaking/media/021515_sUAS_Summary.pdf.
28. Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II," *Lecture notes in computer science.* Vol. 1917, 2000, pp. 849-858.
29. Luke, S. *Essentials of Metaheuristics*: Lulu, 2013.
30. van den Berg, J., Guy, S., Lin, M., and Manocha, D. "Reciprocal n-Body Collision Avoidance," *Robotics Research*. Vol. 70, Springer Berlin Heidelberg, 2011, pp. 3-19.
31. Fiorini, P., and Shiller, Z. "Motion planning in dynamic environments using the relative velocity paradigm," *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*. 1993, pp. 560-565 vol.1.
32. van den Berg, J., Lin, M., and Manocha, D. "Reciprocal Velocity Obstacles for real-time multi-agent navigation," *Proceedings of the 2008 IEEE International Conference on Robotics and Automation*. 2008, pp. 1928-1935.
33. Stroeve, S. H., Blom, H. A., and Bakker, G. B. "Contrasting safety assessments of a runway incursion scenario: event sequence analysis versus multi-agent dynamic risk modelling," *Reliability Engineering & System Safety.* Vol. 109, 2013, pp. 133-149.